Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/GB05/000610

International filing date: 18 February 2005 (18.02.2005)

Document type: Certified copy of priority document

Document details: Country/Office: GB
Number: 0403709.9
Filing date: 19 February 2004 (19.02.2004)

Date of receipt at the International Bureau: 18 April 2005 (18.04.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

PCT/GB 2005 / 0 0 0 6 1 0

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed Andrew Gersey

Dated 5 April 2005

INVESTOR IN PEOPLE

GB 0403709.9

By virtue of a direction given under Section 30 of the Patents Act 1977, the application is proceeding in the name of:

QUALCOMM CAMBRIDGE LIMITED,
Matrix House,
Cambridge Business Park,
Cowley Road,
CAMBRIDGE,
CB4 0HH,
United Kingdom

Incorporated in the United Kingdom,

[ADP No. 09021254001]

Patents Form 1/77

Patents Act 1977
(as amended)

The Patent Office

1/77

THE PATENT OFFICE
JR
19 FEB 2004
RECEIVED BY FAX

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

19FEB04 E874648-1 D01070
P01/7700 0.00-0403709.9 ACCOUNT CHA

1.	Your reference	P4294.A1/PDW
2.	Patent application number *(The Patent Office will fill in this part)*	19 FEB 2004 0403709.9
3.	Full name, address and postcode of the or of each applicant *(underline all surnames)*	Trigenix Limited Matrix House, Cambridge Business Park Cowley Road CAMBRIDGE CB4 0HH UNITED KINGDOM
	Patents ADP number *(if you know it)*	8720567001
	If the applicant is a corporate body, give the country/state of its incorporation	UNITED KINGDOM
4.	Title of the invention	Mobile Communications Network
5.	Name of your agent *(if you have one)*	DUMMETT COPP
	"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*	25 THE SQUARE MARTLESHAM HEATH IPSWICH IP5 3SL
	Patents ADP number *(if you know it)*	6379001

6.	If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of those earlier applications and *(if you know it)* the or each application number	Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*

7.	If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application	Number of earlier application	Date of filing *(day / month / year)*

8.	Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:* *a) any applicant named in part 3 is not an inventor, or* *b) there is an inventor who is not named as an applicant, or* *c) any named applicant is a corporate body.* *See note (d))*	YES

18/1/05 (1977 ACT) APPLICATION FILED

Patents Form 1/77

Patents Form 1/77

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	
Description	66
Claim(s)	0
Abstract	0
Drawing(s)	6

10. If you are also filing any of the following, state how many against each item.

Priority documents	
Translation of priority documents	
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	
Request for preliminary examination and search *(Patents Form 9/77)*	
Request for substantive examination *(Patents Form 10/77)*	
Any other documents *(please specify)*	

11. I/We request the grant of a patent on the basis of this application.

Signature: Dummett Copp

Date: 19th February 2004

DUMMETT COPP

12. Name and daytime telephone number of person to contact in the United Kingdom: Pete Wilson 01473 660600

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) *If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.*

d) *If you have answered 'Yes' Patents Form 7/77 will need to be filed.*

e) *Once you have filled in the form you must remember to sign and date it.*

f) *For details of the fee and ways to pay please contact the Patent Office.*

DUPLICATE

MOBILE COMMUNICATIONS NETWORK

The present invention relates to a mobile communications network and in particular to the operation of and the user interface for mobile terminals for use with a mobile communications network.

One of the growth areas for mobile network operators and content providers is the provision of ringtones, wallpapers and other multimedia content for mobile telephones and terminals. There is a tension between the needs of mobile network operators and terminal manufacturers to retain control over some aspects of the terminal user interfaces for branding purposes and the needs of users to customise and modify the appearance of their terminals to suit their own needs. The sophisticated software required to provide the desired flexibility and customisation is also in tension with the limited processing power and data storage capacity of typical mobile terminals. The present invention seeks to mitigate these problems.

According to an aspect of the present invention there is provided a method of compiling a user interface for a computing device, the method comprising the steps of: identifying one or more resource identifiers in the code comprising the user interface; generating a series of strings comprising one or more characters; and replacing the or each resource identifiers with a unique string. The or each string may contain numerical and/or alphanumerical characters. The series of strings may comprise a series of consecutive numbers or alphanumerical combinations. In an alternative embodiment, the strings may be assigned randomly.

In a further embodiment, the numbers or alphanumerical combinations may be assigned to reflect the relationship of the resources within a hierarchically file system.

According to an aspect of the present invention, there is provided a method of generating a user interface for a terminal, the method comprising the steps of: generating a plurality of sets of user interface elements; ordering each of the plurality of sets of user interface elements into a hierarchical relationship; and rendering the user interface based on the contents of the plurality of sets of user interface elements wherein if more than one set of user interface elements comprises the same user interface element, the user interface element from the set having the highest position within the hierarchical relationship is selected for rendering.

According to an aspect of the present invention, there is provided a method of generating a user interface, the method comprising the step of: creating a container, the container comprising: code for a user interface; one or more content resources for use in the user interface; and metadata relating to the content resources, the code content resources and metadata being stored as objects within the container. The content resources metadata may be updateable in order to allow changes to be made. The method may comprise the further step of exporting the container to a content publishing component that processes the container contents into a format for transmission to a terminal.

Alternatively, the metadata relating to the content resources may relate to one or more hierarchical classifications, the

hierarchical classification(s) relating to the capabilities of a terminal that may receive the content resources.

According to an aspect of the present invention, there is provided a container file, the container file comprising : code for a user interface; one or more content resources for use in the user interface; and metadata relating to the content resources, the code content resources and metadata being stored as serialised objects within the container. The metadata may comprise data determining access to the code and/or the content resources to prevent unauthorised access.

According to a further aspect of the present invention, there is provided a method of displaying a subset from a plurality of user interface elements, the method comprising the steps of: determining the number of UI elements that can be displayed; selecting a subset comprising that number of UI elements from the plurality of UI elements for display; displaying the subset of UI elements within the UI; updating the contents of the subset in response to a user interaction; and updating the display in accordance with the updated contents of the subset.

According to a yet further aspect of the present invention there is provided a method of displaying a content element selected from a plurality of content elements on a remote terminal, the method comprising the steps of selecting a variable associated with one of the plurality of content elements; generating a location identifier that comprises a component that is uniquely associated with the selected variable; transmitting the location identifier to the server,

the server, in response, transmitting the content element identified by the location identifier to the mobile terminal.

The invention will now be described by way of illustration only and with respect to the accompanying drawings, in which

Figure 1 shows a schematic depiction of a system incorporating the present invention;
Figure 2 depicts in greater detail the structure and operation of server 100;
Figures 3a & 3b show a schematic depiction of how the trig resources may be renamed;
Figure 4 shows a schematic depiction of the software 400 for the mobile terminals 300;
Figure 5 shows a schematic depiction of the content toolset 200; and
Figure 6 shows a schematic depiction of four hierarchical planes

Figure 1 shows a schematic depiction of a system incorporating the present invention. The system comprises server 100, content toolset 200, mobile terminals 300, operational support systems (OSSs) 700, content feeds 500 and user interface (UI) sources 600. In use, the server 100 communicates content data and UI data to the mobile terminals 300, 301, ..., each of which comprise software package 400. The server 100 interfaces with OSSs 700, with the OSSs being those conventionally used to operate mobile networks, for example billing, account management, etc. The server 100 further interfaces with the content toolset 200: the content toolset receives data from UI sources 600, 601, ..., and packages the UI data such that the server can transmit the

packaged UI data to the software packages 400 comprised within the mobile terminals 300. The server receives data from a plurality of content feeds, and this data is processed and packaged such that it can be sent to the software packages 400 or so that the mobile terminals 300 can access the data using the software package 400.

The system can be envisaged as being divided into three separate domains: operator domain 50 comprises the systems and equipment operated by the mobile network operator (MNO); user domain 60 comprises a plurality of mobile terminals and third-party domain 70 comprises the content feeds and UI feeds that may be controlled or operated by a number of different entities.

Figure 2 depicts in greater detail the structure and operation of server 100. Server 100 comprises publishing component 110 and content server component 150. Publishing component comprises database 111, import queue 112, content toolset interface 113, user interface 114 & catalogue 115. In operation, the publishing component receives content from the content toolset at the content toolset interface. The content is presented in the form of a parcel (see below) comprising one or more Trigs and one or more Triglets. A trig is a user interface for a mobile terminal, such as a mobile telephone and a triglet is a data file that can be used to extend or alter a trig. If a parcel comprises more than one trig then one of the Trigs may be a master trig from which the other Trigs are derived.

The publishing component user interface 114 can be used to import a parcel into the database 111, and this process

causes references to each trig and triglet to be loaded into the import queue 114, which may comprise references to a plurality of parcels 210a, 210b,... . The contents of the parcel may be examined using the user interface and the contents of the parcel can be passed to the catalogue.

The MNO may have several publishing domains, for example one for each target server in a number of countries or regions. Each domain is treated in isolation from other domains and has its own publishing scheme describing how objects are to be published onto content servers in both live and staging environments. The publishing component GUI provides several different views to each domain, enabling operators to completely manage the publishing of content. The catalogue comprises references to the Trigs stored in the catalogue and the update channels and feed channels used to transfer content to the various domains. For each domain, the operator uses the publishing component GUI to set up the domain structure and allocate trigs from the catalogue to each domain node. To aid the operator in selecting trigs efficiently, a filter is provided in the catalogue so that only relevant items are shown.

A trig may be allocated to several nodes within a domain. In each case the packaging of the trig on the target server may need to be different e.g. a SIS or CAB file, dependent on the handsets that will be accessing the trigs. The packaging can be controlled using the publishing component GUI.

The update channels may be referenced by trigs to control the delivery of the content. An update channel comprises a URL which is a link to a resource on the associated domain that

comprises a triglet update package. The URL can be polled at predefined intervals and the HTTP GET function used to access the package (it will be readily appreciated that other transport schemes may be used with the present invention, for example SyncML or SMS or cell broadcast for small updates). The triglet update package describes how the trig can be modified, e.g. replacing one or more images or text files used by the trig. The publishing component GUI enables an operator to define and control the update channels that exist for a domain, the URLs associated with each triglet on the update channel and the association of triglets with the update channels for a domain. As each triglet is associated with an update channel, an operator may enter the date and time that the update should be published, enabling a schedule to be set.

A content feed is similar to an update channel for which the content updates are automatically generated on a regular basis. A content feed is accessed by polling a URL, retrieving the update packet and applying it to the trig. However because of the different nature of manually constructed triglet updates and automatically generated content, update channels and content feeds are managed separately. Again, other transport schemes may be used such as SyncML or OMA-DM (Open Mobile Alliance Device Management).

The publishing component GUI enables the operator to define which content feeds should be available within each domain and a platform specific location identifier, for example a URL, to which the content should be posted. The operator defines content feeds themselves using a separate screen within the publishing component GUI. All domain publishing

scheme information at this stage is held in the database.

If an existing live Domain is to be modified, the publishing component operates on a copy of the live domain structure, and defines the changes to be made to the domain e.g. the removal or addition of trigs. The individual trigs or triglets allocated to the publishing scheme reference the parcels from which they were originally imported. This enables the publishing component to compile them later (see below).

The operator is able to select different views on the Domain under development, for example the original structure (i.e. what is currently live), a final proposed structure (Approved and/or Pending items), with or without changes marked, the changes only and rejected items. The publishing component GUI prevents a domain scheme from being published if it contains trigs or triglets which reference update channels or content feeds that have not yet been allocated to the domain. Once a publishing scheme is ready to be tested, it is published to the domain's staging server for testing.

The publishing request is processed by the server and comprises compiling all uncompiled Trigs and Triglets (both Approved and Pending) and exporting all proposed changes, both pending and approved, to the Staging Server (this includes new trigs, updates to existing trigs, triglets overdue for publishing (according to a test date) and removal of trigs, triglets and nodes. If there are any failures at compilation stage, no items will be published. The offending item must be rejected or corrected to allow publishing to continue.

Once on the MNO's staging server, the Domain can be tested using mobile devices. Each item that passes its test can be marked as "Approved" using the publishing component GUI. Items that do not pass tests can be marked as "Rejected". Corrected Trigs/Triglets can be imported into publishing component, resubmitted to the Domain and then published to the staging server as described above. This process continues until all items are approved and the Domain is ready for publishing to the live environment. Additionally, for the staging area of a domain, it is possible to simulate the passage of time so that scheduled updates can be tested. Some MNOs may not need the Staging server capability and thus all items to be published can be marked as approved when the domain scheme is set up and thus the operator can proceed directly to live publishing.

The publishing component GUI provides views of each of the domains, for both Live and Staging areas. From this view it is possible to start and stop automatic publishing of content feeds and scheduled publishing of triglets on each update channel.

Having completed testing, the domain may be published to the live area of the server, using a process similar to that described above, except that only domain changes marked as approved are published. When setting up a publishing scheme, the dates and times at which to publish individual Trigs and Triglets may be set. On requesting that a domain is ready for publishing, publishing component ensures that all trig and Triglets are compiled – even if the publishing date or time associated with the item is in the future. Future dated items

remain in the publishing component database until they are ready to be dispatched to the target domain area. A scheduled publish process may poll each target Domain area at configured intervals and dispatches any scheduled items which are due or overdue for publishing to the appropriate Domain target via the dispatch API.

Content feeds are updated at regular intervals with dynamic content being extracted from external sources e.g. web scraping, RSS news feeds, etc. The dynamic content may be simple news ticker text (headlines and URLs) but could also include more complex objects of the type that might be replaced using the content toolbox. The content feed process formats this content into an update file and then passed back to Feed Control.

Feed Control invokes the Compile Trigs process which passes the triglet Parcel template associated with the Content Feed and the update file to the compiler. The compiler extracts the resources in the triglet parcel and returns a compiled triglet to Feed Control which can then be published (see above).

For each trig or triglet to be compiled, the complier requires the following information: the original parcel in which the trig/triglet was imported or created; the list of trig/triglet update packages to be created; the type of files to be created; and a URL map for the update channels and content feeds. The compiler uses the URL map to update the URLs referenced by the update channels (and content feeds) within the individual trigs and triglets within each parcel.

One of the limitations of mobile terminals is that resources which are abundant in conventional computing devices (storage, RAM, processor power, etc.) are scarce and must thus be used in a much more efficient method. In a preferred embodiment of the present invention, when a trig is complied the various resources can be renamed in order to provide a resource path that is shorter than the equivalent filepath for the various resources used in the trig. Figure 3a shows a schematic depiction of how the trig resources may be named prior to compilation and Figure 3b shows an example of how this can be renamed. The shorter filenames mean that the complied trig occupies less storage space on a device and will be transmitted to a terminal over the network more quickly. It will be understood that other naming/numbering conventions may be used, for example alphanumerical, alpha characters only, hexadecimal; etc.. Instead of assigning numbers in a consecutive manner, the numbers may be chosen randomly from a pre-selected range of numbers. Furthermore, it is also possible to us the scheme to describe the hierarchical relationship of the resources within the file systems, for example the content sub-directory could be named 1.1, the images sub-directory 1.1.1, etc. The information regarding the generation of the resource paths is transmitted to the terminal such that the correct resources can be accessed when required. If a trig or triglet is updated then the UI update packet will use resource path references to make the required changes.

This technique may be applied to trigs, triglets and updates. During compilation all references to the resources should be updated in order to ensure that the resources can be accessed. In the case where periodic data updates are

supplied, it may be that a trig refers to content that is not yet present and will be subsequently provisioned using a further triglet. In such a case the compilation process should enable the inclusion of a reference to content that is not present in a trig or triglet. This update should be performed prior to the trig being moved to the content server component 150. It will be readily apparent that such techniques may also be applied to other types of computing devices that do not have the resource limitation of mobile terminals.

A dispatch API may be used for dispatching content to the MNO's servers. A reference FTP model has been implemented to service the API and transfer files to a content server however the API mechanism enables an integrator to implement their own content dispatch mechanism for example using the publishing component output as input to the API of their own content management system, adding custom logic if required.

The publishing component supports conventional OSS functionality accessible via the publishing component GUI and via an industry standard API (JMX) which enables SIs to use standard integration tools to integrate the publishing component into the MNO's OSS environment. This includes the logging of significant publishing component events and all imported/published items, an audit trail for any changes noted with external scripts, maintain Error/Warning Logs, system alerts, health check report, etc. All data relating to the publishing component is stored within a database, such as Oracle, and backup and restore functionality is supported by the standard database processes, integrated with the OSS environment. For operation the publishing component requires

an operating J2EE environment and an installed instance of a database, such as Oracle. Installation of the publishing component can be validated by a process that indicates that that the installation process was successful and that all components have been activated correctly.

The content server component 150 is a standard implementation of a web server and as such the scaling model is well understood. The capabilities of a server can be rated using a "SPECweb99" number indicating the number of concurrent sessions that the web server can handle under benchmark conditions. Published SPECweb99 numbers range from 404 to 21,000 with typical commercial web servers having SPECweb99 numbers in the order of 5,000. A typical deployment scenario for the present invention of 1m subscribers with hourly updating content requires a web server with a SPECweb99 rating of only 1,112. A successful deployment of the present invention will lead to increased service use which can be provided for by enabling additional servers to create an infrastructure that can be both scalable and highly resilient to failure.

A connection may be made to the server from a mobile terminal via a WAP Gateway. In this case the web server session exists between the WAP gateway and the web server, rather than the mobile phone and web server. When a request is made for a file via the WAP gateway, the session with the web server lasts only as long as it takes to transfer the file from the web server to the WAP gateway - i.e. the session is extremely short since the connection bandwidth will be very high and latency extremely low.

Alternatively a direct connection may be established between the mobile phone and the web server. In this case, the web server will need to keep the session open for as long as it takes to download the data to the phone.

There are two types of content that are delivered by the content server component: trigs, typically of the order of 100KB and regularly updating triglets which are typically of the order of 1KB. The traffic created by trig downloads is very similar to the traffic generated by existing content downloads. And thus the related issues are well understood. Downloads of regular triglet updates are a new feature in an MNO's traffic model but because of the small size of the updates, which typically fit within one data packet, it is possible to show that the traffic can still be handled by typical web servers.

In the case of a triglet download, typically only one data packet is required to transfer 1KB. Assuming a round-trip latency across a GPRS network of 2 seconds, the web server will need to hold open a typical session for around 4 seconds. For the scenario of 1 million subscribers having a trig on their phone with content that updates every hour, this implies 278 hits per second on the web server and 1,112 concurrent sessions. As stated earlier, this number is well within the capability of typical web servers.

Figure 4 shows a schematic depiction of the software 400 for the mobile terminals 300, which comprises a mark-up language renderer 410, update manager 420, network communication agent 425, resource manager 430, virtual file system 435, actor manager 440, a plurality of actors 445a, 445, ..., native UI

renderer 450, support manager 460, trig manager 465 and mark-up language parser 470.

It is preferred that the software operates using TrigML, which is an XML application and that mark-up language renderer 410 renders the TrigXML code for display on the mobile terminal 300. The mark-up language renderer also uses the TrigML Parser to parse TrigML resources, display content on the terminal screen and controlling the replacement and viewing of content on the handset. The native UI renderer is used to display UI components that can be displayed without the use of TrigML, and for displaying error messages.

The software 400 is provisioned and installed in a device specific manner. For example for a Nokia Series 60 terminal the software is installed using a SIS file, whereas for a MS Smartphone terminal the software is installed using a CAB file. Similarly, software upgrades are handled in a device specific manner. The software may be provisioned in a more limited format, as a self-contained application that renders its built in content only: i.e. the software is provisioned with a built-in trig but additional trigs cannot be added later. The supplied trig may be upgraded over the air.

The trig manager 465 presents an interface to the resource manager 430 and the mark-up language renderer. It is responsible for trig management in general. This includes: persisting knowledge of the trig in use, changing the current trig, selection of a trig on start-up, selection of a further trig as a fall back for a corrupt trig, maintaining the set of installed trigs, identifying where a particular trig is installed to the resource manager and reading the update

channel definitions of a trig and configuring the update manager appropriately.

The resource manager provides an abstraction of the persistent store on device, i.e. storing the files as real files, or as records in a database. The resource manager presents a file system interface to the mark-up language renderer and the update manager. It is responsible for handling file path logic, distinguishing between real resource files and actor attributes, mapping trig-relative paths onto absolute paths, interfacing with the trig manager and providing a modification interface to the update manager.

The Resource Manager is also responsible for ensuring the integrity of the resources stored in the persistent store, especially in the face of unpredictable interruptions such as loss of device power. The Resource Manager has no knowledge of the trig currently used. Its interface is thread safe (as it may be used by both the Update Manager and the Renderer from different threads.

The Update Manager handles the reception and application of Trigs and Triglets. The Update Manager presents an interface to the Renderer and the trig Manager and is responsible for: the initiation of manual updates when instructed to by the Renderer; controlling and implementing the automatic update channel when so configured by the trig manager; indicating the progress of a manual update and recovering an Update following unexpected loss of network connection and/or device power. The update packet format may be defined as a binary serialisation of an XML schema.

The Support Manager provides an interface for other components to report the occurrence of an event or error. Depending on the severity of the error, the Support Manager will log the event and/or put up an error message popup

XML is a convenient data formatting language that is used to define the update packet format as well as TrigML content. For bandwidth and storage efficiency reasons, text XML is serialised into a binary representation. Both update packets and TrigML fragments are parsed by the same component, the MARK-UP LANGUAGE PARSER parser. Any further use of XML in the software must use the binary XML encoding and therefore re-use the parser.

The Actor Manager 440 looks after the set of actors 445 present in the software. It is used by: the renderer when content is sending events to an actor; actors that want to notify that an attribute value has changed and actors that want to emit an event (see below).

In a preferred embodiment, the software comprises a multi-threaded application running a minimum of two threads, with more possible depending on how many and what sort of actors are included. The software runs mostly in one thread, referred to as the main thread. The main thread is used to run the renderer which communicates synchronously with other components. Actors always have a synchronous interface to the Renderer. If an actor requires additional threads for its functionality, then it is the responsibility of the Actor to manage the inter-thread communication. It is preferred that a light messaging framework is used to avoid unnecessary code duplication where many actors require inter-thread

communication.

In addition to the main thread, the update manager runs a network thread. The network thread is used to download update packets and is separate from the main thread to allow the renderer to continue unaffected until the packet has arrived. The Update Manager is responsible for handling inter-thread messaging such that the Update Manager communicates synchronously with the Renderer and Resource Manager when applying the changes defined in an Update Packet.

The memory allocation strategy of the software is platform specific. On MIDP platforms, the software simply uses the system heap and garbage collector for all its memory requirements. Garbage collection is forced whenever a content replacement event occurs in an attempt to keep the garbage collection predictable and not suffer unexpected pauses in operation. It is assumed that any memory allocation might fail, in which case the software will delete all its references to objects, garbage collect, and restart - assuming that the software has already successfully started up and rendered the first page.

On C++-based platforms, a mixture of pre-allocation and on-demand allocation will be made from the system heap. All memory required for start-up is allocated on-demand during start-up, with any failures here causing the exit (with message if possible) of the software. Following successful start-up, memory needed for rendering the content document model is pre-allocated. Provided content is authored to use less than a defined limit, it is guaranteed to render.

Additional use is made of RAM for various caches needed for fast operation of the software. Where memory conditions are low, these caches will be released resulting in slow rendering performance from the software.

Errors that are severe enough to disrupt the normal operation of the software must result in a pop-up dialog box. The dialog box contains one of a small number of internationalised error messages (internationalised versions of these strings may be compiled into the software at build-time with the version of an error string to display being determined by the relevant language setting on the device). To keep the number of messages to a minimum, only a few generic problems are covered.

To allow for support situations, error dialogs also display an error code as a 4-digit (16-bit) hex string. Each error code is associated with a description text that can be used by support staff to determine the nature of a problem with the software. Errors that occur in the software and that are not severe enough to halt its operation may be logged by the Support Manager component. The Support Manager can be queried by the user typing special key sequences. The Support Manager can also supply its error log to a server via an HTTP GET or POST method.

The Renderer receives information regarding the key press. If there is no behaviour configured at build time for a key, it is sent as a TrigML content event to the current focus element. The content event is then handled as defined by TrigML's normal event processing logic.

For example, if a key is pressed down, a 'keypress' event is delivered to the Renderer with a parameter set to they relevant key. When the key is released, a '!keypress' event is delivered to the Renderer. If a key is held down for a extended period of time, a 'longkeypress' event is delivered to the renderer. On release, both a '!longkeypress' and a '!keypress' event are delivered to the Renderer.

Whenever the software is started, it executes the following actions:

- Check for, and continue with, interrupted Update processing;
- Check for, and process, Updates residing in the file system (either pre-provisioned, or installed to the file system by some other means);
- If known, start the current trig (which may be the last run trig);
- If a current trig is not set, a trig that has been flagged as a 'default' trig can be started.
- Failing the presence of a default trig, the first valid trig by alphabetical order of name will be selected.

A trig is started by loading the defined resource name, start-up/default. The TrigML defined in start-up/default is parsed as the new contents for the content root node.

The first time a trig is run by the software following its installation, the trig is started by loading the resource name startup/firsttime. The software may record whether a trig has been run or not in a file located in the top level folder for that trig. Dependent on the platform used by the mobile terminal, the automatic start-up of the software may

be set as a build-time configuration option. Furthermore, placing the software in the background following an auto-start may also be a build-time configuration option.

A launcher may appear to the user as an application icon and selecting it starts the software with a trig specified by that launcher (this trig may be indicated by a launcher icon and/or name). When using a launcher to start a trig, it is possible to specify an 'entry point' parameter. The parameter is a resource name of a file found in the 'start-up' folder. This file is not used if the trig has never been run before, in which case the file called 'firsttime' is used instead.

The software uses content resource files stored in a virtual file system on the device. The file system is described as virtual as it may not be implemented as a classical file-system, however, all references to resources are file paths as if stored in a hierarchical system of folders and files.

Details regarding the arrangement of the file-system for a preferred embodiment of the present invention are given below in Appendix A. Furthermore, the software stores some or all of the following information: usage statistics; active user counts; TrigManager state; TrigML fragments & update channel definition (serialised as binary XML); PNG images; plain text, encoded as UTF-8 OTA and then stored in a platform specific encoding; other platform specific resources, e.g. ring tone files, background images, etc.

Files in the file system can be changed, either when an actor attribute value changes, or when a file is replaced by a triglet. When files in the /attrs directory change, the

Renderer is immediately notified and the relevant branches of the content tree are updated and refreshed. When images and text resources are changed, the Renderer behaves as if the affected resources are immediately reloaded (either the whole content tree or just the affected branches may be refreshed). When TrigML fragments are changed, the Renderer behaves as if it is not notified and continues to display its current, possibly out of date, content. This is to avoid the software needing to persist <include> elements and the <load> history of the current content.

The software 400 is provisioned to mobile terminals in a device specific method. One or more Trigs can be provisioned as part of the installation, for example, stored as an uncompressed update packet. On start-up, the packet can be expanded and installed to the file-system.

The Actors 445 are components that publish attribute values and handle and emit events. Actors communicate with the Renderer synchronously. If an actor needs asynchronous behaviour, then it is the responsibility of the actor to manage and communicate with a thread external to the main thread of the Renderer.

Actor attributes may be read as file references. Attributes are one of four types: a single simple value; a vector of simple values; a single structure of fields, each field having a simple value; or a vector of structures. Attributes may be referenced by an expression using an object.member notation similar to many object-orientated programming languages:

<image res="signallevels/{protocol.signalstrength}"/>

When needed as a file, an attribute is accessed via the /attrs folder.

<text res="/attr/network/name">

An Actor can be messaged by sending it an event with the <throw> element. Events emitted by actors can be delivered to the content tree as content events: these can be targeted at an element Id or 'top'. The interface to an actor is defined by an Actor Interface Definition file. This is an XML document that defines the attributes, types, fieldnames, events-in and parameters, and events out. The set of actors is configurable at build-time for the software. Appendix B gives an exemplary listing of some actors that may be used, along with the associated functions or variables.

One of the limitations that is common in most mobile terminals is that the display screen is quite small and when a menu is displayed it is not always possible to display all of the menu items on the screen at one time. Conventional approaches tend to load all of the menu items into memory, along with associated icons or graphics, and then display them appropriately as the user scrolls up or down the menu.

The present invention provides a technique that limits the number of menu items to be loaded into memory to the number of items that can be displayed on the screen at a time. When the user scrolls along the menu, the item(s) no longer on display are discarded and the item(s) now on display are loaded into memory.

Preferably, this can be implemented by using a <griddata> element in TrigML to define a list view of some data, where the data is stored in a folder in the file system, and the list appearance has the same structure for each item. The <griddata> element comprises a 'repeat-over' attribute that specifies the folder in which the data can be located. The single child element of <griddata> is a template for the appearance of each item in the list.

The template uses a special symbol, e.g. '$$' to refer to the iterator. This is the template variable that changes each time the template is instantiated: for example

```
<griddata repeatover="news/headlines">
     <text res="news/headlines/$$/title.txt"/>
</griddata>
```

where the folder news/headlines/ contains:

```
0/title.txt
1/title.txt
2/title.txt
3/title.txt
```

This would display a list of 4 items, each described by a simple <text> element pointing the 'title.txt' resource in the 'news/headlines/$$' folder. Where the source data has more items in it than the <griddata> element has room for on the display, the <griddata> element only displays those items that can be displayed. When the user scrolls through the list, the <griddata> element shifts the 'data-window' accordingly. The advantage of this technique is that only

the resources required by the current display are actually loaded in memory, which reduces the memory utilisation and reduces the amount of time taken to render the list of items.

A similar scheme can be used to define the order that a list is displayed in. If the target of the 'repeat-over' attribute is a file instead of a folder, then the file can be assumed to contain a list of resource names to use in the iteration. For example,

```
<griddata repeatover="football/league">
    <text res="football/teams/$$/name.txt"/>
</griddata>
```

where the file football/league contains:

```
Manchester
Arsenal
Chelsea
```

the folder football/teams/ contains:

```
Manchester/name.txt
Arsenal/name.txt
Chelsean/name.txt
```

and each name.txt is a text file holding the team name. The result of this is that the test files associated with the teams would be displayed in the defined order and within the defined area of the terminal display.

Where data is accessed by means other than the file system, e.g. it is stored in a database, or it is generated on the fly by another software component, this scheme can still be used if a virtual file system 435 is used, which can map a

file system interface onto the underlying provider of the data. This means the content can still be arranged as described above, but the data can be provided in a method that enables efficient data storage and retrieval.

For commercial reasons it is desirable for MNOs and/or content providers to be able to have some control over the user interface that will be displayed on the screen of a mobile terminal. It is also important that there is a degree of flexibility that allows users to download triglets or new trigs to modify the appearance of their terminal and also to make further changes to the displayed image that is determined by the trig or triglet in use.

According to an embodiment of the present invention this problem is addressed by considering the UI to be formed from a number of hierarchical planes, each of the levels comprising one or more entities of the UI. By assigning a hierarchy to the MNO, terminal manufacturer, trig provider and the terminal user it is possible to provide the required levels of permission.

Figure 6 shows a schematic depiction of four hierarchical planes 405a-d: plane 405a comprises UI elements defined buy the MNO; plane 405b comprises UI elements defined buy the terminal manufacturer; plane 405c comprises UI elements defined by a trig; and plane 405d comprises UI elements defined buy the user. Plane 405a has the highest position in the hierarchy and p[lane 405d has the lowest position in the hierarchy. For example, the mno_logo element in plane 405a defines the graphic element used and its position on the display screen of the terminal. As it is in the highest plane

of the hierarchy it will always appear and will take preference over any other UI element in a lower hierarchy element that attempts to use the pixels used by mmo_logo. Plane 405d comprises the backgroundcolour element, which is not defined in any of the other planes and thus the colour defined in backgroundcolour will be used in the UI.

Plane 405c comprises the windowtitle.txt element that defines the attributes for the text used in the title of a window. This may be overwritten by adding a windowtitle.txt element to either plane 405a or 405b to define the text attributes, or by adding a windowtitle.txt_deleted element to either plane 405a or 405b to instruct the UI renderer to ignore any subsequent windowtitle.txt element.

In am embodiment of the present invention, the user can set a preference within the software 300 to control the content that is displayed within the UI. For example, content relating to a number of football teams may be stored on a server with a path having a form similar to

/demoUI/football/team_xxxx/team_menu.png

where the team_xxxx variable is selected by the user from a list of teams (manu, chel, leed, manc, etc.) and inserted into the path such that the UI displays the content related to the selected team. A change in the team_xxxx variable will cause the content displayed to altered accordingly. It should be noted that the selection of a preference controls the display of content that is selected from content stored on a remote server, as opposed to selecting from content that is stored on local storage.

This approach is preferable to sending a request of the form

http://t1.trigenix.com/triglets/football/triglet&pn="07766554
43322"

as in this case the server needs to perform a database query in order to identify the content to be displayed and this will significantly increase the resources required from the server to provide the requested content.

Another known technique by which the same result can be achieved is to send a request of the form:

http://t1.trigenix.com/triglets/football/triglet&fc='ManU'

but a disadvantage of this approach is that each time a new team is added then the server logic must be updated to include the new team. In contrast, the method of the present invention merely requires that content is added to the server at a new location, which is a simpler process and requires fewer resources to implement it.

In conventional mobile terminals, information regarding the battery strength, signal strength, new text messages, etc. are shown to the user. Typically this information is obtained by the operating system sending a call to the relevant hardware or software entity and the UI interpreting the received answer and displaying it.

According to the present invention, this information may be displayed in the UI using a TrigML tag (see below)

<phonestatus> (or <signalstrength>). Rendering this tag causes a listening query to be opened to the relevant hardware or software entity. If a change in state occurs then the UI renderer is notified and the UI renderer loads the relevant icon or graphic to communicate the change in state to the user. If the user changes the view within the UI the tag may be withdrawn and the listening query is terminated. This approach is more resource efficient as the listening query is only active when the tag is in use.

Updates comprise a new trig (a new or replacement UI) or a triglet (a modification to an existing trig) and may be regarded as modifications to the software file-system. The Update Manager to determine what needs changing in the file-system by reading a packet. Update Packets may be downloaded over the air by the software 400 using HTTP, or other suitable transport mechanisms, wrapped in a device-specific package format or pre-provisioned with the installation of the software itself.

Updates may be triggered by a number of means, which include

- the software checking for interrupted Update processing on start-up
- the software checking for pre-installed Update Packets on start-up
- automatically as configured by an Update Channel
- user initiation
- the terminal receiving a special SMS

Updates sent OTA can be fetched using HTTP-GET requests initiated by the software. The GET request is directed at the URL associated with the Update. The body of the HTTP response

is a binary file carrying data in an Update Packet format. The data reception is handled in a separate thread to the Renderer thread. For background updates (automatically initiated) this allows the user to continue navigating the UI. For foreground updates (manually initiated) this allows the Renderer thread to display a progress bar and listen for a cancel instruction.

Trigs may define Update Channels, which comprise a URL from which to poll for updates and a number of associated properties: automatic or manual updating; a timing scheme and retry strategy in the event of failure. The software may only initiate an automatic update when it is running.

On start-up, if an update event is due the software will wait for a time interval before beginning the Update. This is to postpone any start-up delays incurred by initiating an Update immediately, and will therefore give the user a faster start-up. The update channels may be defined in a well known location within the trig, for example the config/channels folder of a trig in files containing <channel> tags.

The algorithm used to unpack and install an update is device specific. However, it is important that the algorithm is safe from unexpected interruption (e.g. power loss), such that no corruption or unrecoverable state is reached in the file-system. This may be achieved by using two threads (a network thread and a renderer thread) with the goal of having as much of the update processing as possible being performed by the network thread so as to interrupt the renderer thread for the shortest possible amount of time.

There are other failure modes to consider: if an HTTP-GET cannot be initiated, or is met with an HTTP error response code, then this attempt at an Update is abandoned and the retry strategy is used to begin a new update attempt at a later date. Where an HTTP response is interrupted by loss of network signal, any temporary files are deleted and the retry strategy is used to restart the Update attempt at a later date. If an update header indicates that the update payload size may be too great to fit on the device, if the update requires an incompatible version of the software or if the Update already resides on the device then the header data file is deleted and the Update attempt and any subsequent retries are cancelled.

The content format is common across all platforms implementing the software. The Content Compiler is a content authoring tool to transform a collection of raw resources (text TrigML, PNG images, text string definitions) into an over the air Update Packet that can be written to the file system of the terminal.

TrigML fragments are files containing text TrigML and resource references inside these fragments are virtual file paths. The mapping of these virtual file paths to real file paths is defined by a TrigDefinition file. This file also defines other properties of the trig. When used for compiling a triglet, this file also defines how the input TrigML/PNG/Text resources map onto modifications of the virtual file-system of a trig.

For PNG and Text Resources the trig Definition file points at a list of real files on the host file-system and the

resources are copied to the outputs.

TrigML can use constant variables instead of attribute values. Constant variables are accessed with the same syntax as <include> parameters, e.g. $background_colour. Constants are treated as global variables in a trig and are defined in the reserved folder, constants/. The variable definitions contained in the files in the constants/ folder may be resolved at compile time with direct substitution of their values. In an alternative embodiment the variable definitions in constants/ are compiled as global variables and resolved at content parse time by the software. This allows the trig to be updated by a simple replacement of one or all of its constants files.

A System String Dictionary defines the integer values to use for all well known strings, i.e. reserved words. These have several types, including: TrigML element and attribute names ('group', 'res', 'layer', 'image', 'x'), TrigML attribute values (e.g.: 'left', 'activate', 'focus') and common resource paths (e.g.: 'attr', 'start-up', 'default'). As an input, the String Dictionary is optional. The first time a trig is compiled it will not have a String Dictionary. This first compilation creates the String Dictionary, which is then used for all future compilations of that trig. Triglet compilation must have a String Dictionary that defines all the string mappings used by the trig it is modifying.

An OTA Update Packet comprises one or two files, depending on whether the update is defined with an inline payload or not, and this is determined by the trig Definition file. This package can be placed on a server ready for access by a

mobile terminal or it can be included in the installation package for the mobile terminal software.

In order to successfully render the user interface of a mobile terminal, the mark-up language must have the following qualities: concise page definitions, consistent layout rules, be implementable in a compact renderer, provide multiple layering and arbitrary overlapping content, event model, require the repaint of only the areas of the display that have to change between pages of the UI, include hooks to the platform for reading property values receiving events and sending events, extensible, and be graphically flexible. TrigML provides these features and Appendix C gives an overview of the elements and attributes that provide the desired functionality.

It is desirable that the cost of re-branding UIs and producing a continual stream of updates is minimal. The present invention enables this by providing an efficient flow of information from the creative process through to the transmission of data to users.

A container, referred to as a parcel, is used for UIs, UI updates, and templates for 3rd party involvement. Parcels contain all the information necessary for a 3rd party to produce, test and deliver branded UIs and updates. Figure 5 shows a schematic depiction of the content toolset 200, which comprises scripting environment 220, test and simulation environment 230 and maintenance environment 240

The parcel process comprise five processing stages:

1) The scripting environment 220 provides the means to design

the template for one or more UIs and the update strategy for UIs based on that template.

2) The maintenance environment 240 provides for rapid UI and update production in a well-controlled and guided environment that can be outsourced to content providers.

3) The maintenance environment 240 'pre-flight' functionality allows the deployment administrator to check and tune the UIs and updates that they receive from 3rd parties.

4) The publishing component 110 provides management of UIs and updates at the deployment point, including the staging of new releases.

5) The publishing component 110 enables the automatic generation of updates from live content feeds.

In a typical project, parcels are created within the scripting environment 220 for: a content provider to create re-branded UIs from a template, incorporating the same 'feel' but a different 'look'; a content provider to create updates from a template, that provide a periodic, or user selected variation to UI content ; or an ad agency to create updates from a template that promote new services on a periodic basis.

For all of these use cases, maintenance environment 240 is used to import the parcel, re-brand and reconfigure the content and create a new parcel for submission to the publishing component 110. In the design of the UI template, the following issues should be considered: which part of the UI can be re-banded; which features of a UI need to be reconfigured at re-branding or remotely; which part of the UI content may be updated; and if the UI is re-branded then can user select content feeds in use. The scripting environment

220 allows these strategies to be defined, and enables the maintenance environment 240 as the implementer of each instance of each strategy.

The main functions of the scripting environment 220 may comprise:

- Defines menu structure and page map.
- Defines the framework into which branding content is placed.
- Defines the parts of the UI that are updateable.
- Defines the parts of updates that are replaceable for re-branding.
- Provides an interactive preview to assist editors
- Provides a graphical code view of each UI layer
- Allows drag and drop of resources into the interactive preview and code view.
- Exports templates for specific re-branding or update construction tasks
- Simulates UIs and updates on handset simulator.
- Builds UIs and updates for testing on the real device.
- Provides extended debugging tools to aid development.

Furthermore, the purpose of the maintenance environment 240 is to provide a designer and administrator's UI for the re-branding and maintenance of skins and updates, with the main functions comprising:

- Re-branding UI templates
- Populate updates with new content
- Manage UI menu entries via updates
- Translate UIs and updates for additional languages
- Purpose strings and content for additional devices

- Simulation of UIs deployed on handset simulator.
- Build of UIs and updates for testing on a real device

Rebranded UI

A parcel is generated by the scripting environment 220 which comprises a template UI or update for editing. Once editing is complete the parcel is saved in an 'outbox' ready for despatch to the maintenance environment 240 for publishing to the content server. The following 'parcel' functions are provided. The maintenance environment 240 can be used to edit/replace resources held within the parcel. Parcels can be exported to the simulation environment to test the performance of the UI or UI update on a mobile terminal.

An explorer is provided for the user to access these categories, with the user being able to change: any UIs or updates marked as visible or the resources within a UI or update that are marked as 'replaceable'. When saved, a thumbnail of the 'visible' object is saved in the parcel, for identification use in the maintenance environment and for other services.

A parcel entry may be double clicked to launch an appropriate editor. (for example, an image resource would launch an image editor). All resources may have a text description/note inserted in the maintenance environment and displayed in the appropriate context in the maintenance environment. Lists of menu entries are handled as a special resource type with each entry presenting its own sub-catalogue of resources (for example - title, help string, image, roll-over image, URL and ringtone preview).

Many different UIs can be derived from a common base. Typically the common base would implement most of the interface itself, and Trigs derived from it would implement small variations on it, such as branding. A Triglet can be derived from a Trig, and it can override any of the resources from the parent Trig that it chooses to (optionally it may introduce its own resources). Note that "resources" here also refers to TrigML, so the behaviour and layout of a Trig can be modified by a Triglet just as easily as it replacing a single image or piece of text.

A Parcel may comprise one or more base Trigs (i.e. a Trig that is not derived from any other trig), one or more multiple Trigs derived from a base Trig, a plurality of triglets derived from any of the trigs and a plurality of triglets derived from other triglets.

The parcel format is an opaque binary format that stores all this information as serialized objects. The parcel may comprise a number of resources , such as images, text, URLs, update channels, ringtone files, wallpapers, native applications, etc. Each resource contains permission information as to how to view, edit, or delete the resource. Each resource furthermore contains meta information such as documentation and instructions that are relevant to that resource. Each Parcel tool either assumes a relevant role, or requires users to login as a particular role.

The nature of developing trigs is such that a number of people and/or groups of people could be involved in contributing to the final design and implementation of a Trig. Furthermore, the skill sets of these people require

that a very simplified and controlled scheme be used to minimize the risk of unwitting damage to the Trig. A typical development workflow for a reasonably complex Trig could comprise:

- UI Designers create the original UI structure. This design may be built using the maintenance environment to create the first versions of this UI.
- A graphics designer creates the final graphics, and adds them to the design.
- The areas of the UI dedicated to dynamic content that were identified in the original design need to be fleshed out.
- Graphics for the dynamic update need to be finalised by a graphics designer.
- Personalisation areas of the UI are then designed and implemented. This might be handled by a number of third-party content providers.

Parcels assist in the workflow described above because they contain the entire project in the single file, and this makes it easy to pass from one member of the team to the next. The Parcel can be re-targeted for the next stage of development by adding comments and instructions on the resources that need to be modified, and even setting the editability of other resources to restrict what can be changed. More complicated workflows can be supported by allowing Parcels to be forked, and separate development to happen in each fork of the Parcel. Merge tools allow the individual changes to be combined back into a single Parcel. A parcel may be implemented using the pickle module for the Python programming language.

The parcels may be used to develop trigs and/or triglets for mobile terminals having different capabilities such as display size, RAM capacity. To simplify this, a number of hierarchies may be defined and the data resource or TrigML element classified within the hierarchies. For example, a hierarchy of terminals may be:

Nokia 7650 > Symbian S60 > Defined Screen Size > Any device

and the language used in the resource may be classified in a two-level hierarchy of language specific > language independent. When a trig or triglet is compiled from a parcel, the most appropriate resources or TrigML elements can be selected and complied for a particular terminal.

APPENDIX A

For file paths beginning with a leading '/':

/attrs	Like the unix /proc directory, stores actor attribute values for reference by content when the attribute is needed as a file reference.
<actor>	Each subdirectory of /attrs is the actor name.
<attribute>	Each attribute is accessed as a node in the actor subdirectory
<field>	If the attribute is a structure, then the field name specifies which structure member to access.
<index>	If the attribute is a vector attribute, then the index number specifies the index into the vector of the desired attribute.
<field>	If the vector attribute is a collection of structures, then the field name again specifies the structure member.

File paths without a leading '/' are treated as relative to the current trig, i.e. every trig is stored in its own folder hierarchy rooted in a single folder.

config	Common folder in every trig to store trig meta data.
channels	Common folder to store the update channel definitions.
<channel defs>	Set of files defining the collection of update channels for the trig. Each file can define one or more update channels.
start-up	Common folder to store entry points for the trig.
default	Common TrigML file to store the default entry point for the trig.
firsttime	Common TrigML file to store the TrigML for use the first time this trig is run
<trigml files>	Other named TrigML files can be used as entry points if found in the start-up folder.
constants	This folder is not passed OTA and is instead fully resolved at content compile time.
<rest of content>	trig content is organised in trig-defined format under the Trigs folder.

APPENDIX B

Trigplayer Actor	Attributes	UpdateState	
	Messages	exit	
		predial_mode	on/off
	Events	idle	

Launch Actor	Attributes		
	Messages	browser	url
		SMS	Number
			message
		Camera	
		Inbox	
		Profiles	
		missed_calls	
		dialer	number
		""	
		native_app	app_id
			url
	Events		

Install Actor	Attributes		
	Messages	ringtone	resource_path
		wallpaper	resource_path
	Events		

Phone Actor	Attributes	Bluetooth	
		IrDA	
		Call	
		GPRS	
		UnreadSMS	
		UnreadVoiceMail	
		UnreadMsgs	
		BatteryLevel	
		SignalStrength	
	Messages		
	Events	missed_call	
		message_arrived	
		voice_mail_arrived	

APPENDIX C

<trigml>

<layer>
id

Visible Elements

common attributes
id x y w h bdcolor
bgcolor
hasfocus canfocus clip
raise

<group>

<grid>
rows cols rowsplit
colsplit

<griddata>
repeatover
rows cols rowsplit
colsplit

<gridlist>
initrow initcol
rows cols rowsplit
colsplit

<image>
res frames index

<tile>
res bdt bdb bdr bdl

<text>
res font size slant
weight align
color fxcolor multiline

<paintif>
res isvalid

<ticker>
repeatover

Listener Elements

common attributes
when consume

<throw>
event target

<att>
name value valuefrom

<anim>
name duration repeat
persist startvalue
endvalue bounce

<load>
res target

<setvar>
name value valuefrom

System Events

entry
focus **!focus**
keypress [key]
!keypress [key]
longkeypress [key]
!longkeypress [key]
moreUpChanged [newValue]
moreDownChanged [newValue]

<batterylevel> res frames **<signalstrength>** res frames **<phonestatus>** res include **<include>** res **<param>** name value valuefrom	

type: visible			Class of element that can have a visible representation on the display. This section describes attributes and properties common to all *visible* elements.
contains	contained by		
any *listener*	any *container*		
attributes	type	default	
id	*string*	none	The name or ID of this element. This identifier is used in the target attribute of <throw> and <load> elements. If the same ID is used more than once, then the last ID loaded is used.
x *(modifiable)*	*integer* left centre center right	centre	The x-coordinate of the frame of the element, relative to the top-left corner of the parent element. If one of 'left', 'centre' or 'right', the frame is suitably aligned within parent element.

y *(modifiable)*	*integer* top centre center bottom	centre	The y-coordinate of the frame of the element, relative to the top-left corner of the parent element. If one of 'top', 'centre' or 'bottom', the frame is suitably aligned within parent element.
w *(modifiable)*	*integer* *	*	The width of the frame of the element. If '*', the frame assumes the width of the parent frame, or cell, if it is in a grid.
h *(modifiable)*	*integer* *	*	The height of the frame of the element. If '*', the frame assumes the height of the parent frame, or cell, if it is in a grid.
bgcolour, bgcolor *(modifiable)*	*colour*	#00000000 (trans-parent)	The background fill colour of the element. If translucent alpha values are not supported, then the alpha component will round down to fully transparent.
bdcolour, bdcolor *(modifiable)*	*colour*	#00000000 (trans-parent)	The colour of the border for this element. The border is drawn 1-pixel wide and just *inside* the frame. The border can be partially or fully obscured by the child contents. If translucent alpha is not supported, then the alpha component is rounded up to full opacity.
clip	*boolean*	true	If true, the painting of all child contents of this element

			will be clipped by the frame of this element, i.e. children cannot 'spill' outside the frame. If false, the painting of all child contents will be clipped by the clipping frame of the parent element. clip=false should be used with caution as it slows down the renderer.
raise *(modifiable)*	boolean	false	If true, the painting of this element is painted last within its <layer>. If more than one element specifies raise=true, then they are all painted last, but in their normal relative order. If false, the painting of this element is in the normal order - that of painting elements in the order parsed.
hasfocus	boolean	false	If true, this element will be given the initial focus for the layer that it is in. If more than one element specifies hasfocus=true, then the last within each layer to do so is given the initial focus. When loading new content that contains an element with hasfocus=true, the focus is only given to this element if the new content is removing the element that previously had

			the focus.
canfocus	boolean	false	If true, this element will be given the focus when navigating with the cursor keys. If false, this element will be ignored when navigating with the cursor keys. (Note: This replaces: <att when=focus/>)

<trigml>

The root element of all TrigML documents. It does not have any visual appearance.

contains	contained by
any element	*none*

attributes	type	default
none		

<layer>

Full screen layer. Each layer manages its own focus. The highest layer with a non-null focus element gets keypresses and events sent to _top.

contains	contained by
any *visible* any *listener*	_top

attributes	type	default	
id	*string*	*none*	The name or ID of this element. This identifier is used in the target attribute of <throw> and <load> elements. If the same ID is used more than once, then the last ID loaded is used.

<group>	Generic container of other

visible, container			elements. Can be used as a plain rectangle.
contains	contained by		
any *visible* any *listener*	any *container*		
attributes	type	default	
all attributes in type: *visible*			

<grid> *visible, container*			Container element that arranges its children in a grid. <grid> is purely for layout. Use <gridlist> or <griddata> for focus management. Each child is placed in its cell, with that cell forming its parent frame - i.e. children that leave w/h as '*' will be the size of their cell.
contains	contained by		
any *visible* any *listener*	any *container*		
attributes	type	default	
rows	*integer*	*none*	The number of rows in the grid. Cannot be zero. If rows is supplied and cols is not, then the grid is filled column by column.
cols	*integer*	*none*	The number of columns in the grid. Cannot be zero. If cols is supplied and rows is not, then the grid is filled row by row. If both rows and cols are supplied, then the grid is also filled row by row.
rowsplit	*list of semi-*	*	The heights of each row. If fewer values are supplied than

	*colon separated integers or *s*		there are rows, the last value is repeated for each extra row. All rows that have * for a rowsplit share the available space.
colsplit	*same as rowsplit*	*	The column width equivalent of rowsplit.
all attributes in *type: visible*			Note that clip applies to the whole grid, not each cell in the grid.

<gridlist> *visible, container*			Container element that arranges its children in a grid. It is also a Focus Manager in that it moves an active cell around the grid, scrolling the grid if the grid is bigger than the frame of this element. Note that both rows and rowsplit, and cols and colsplit, must be supplied to achieve a grid that is larger than the w/h of this element.
contains	contained by		
any *visible* any *listener*	any *container*		
attributes	**type**	**default**	
initrow	*integer*	*0*	The initial row of the active cell. Count from zero. See initcol below.
initcol	*integer*	*0*	The initial column of the active cell. Count from zero. The first time the gridlist gets focus, this is the cell that is in turn given focus. The hasfocus attribute overrides initrow and initcol.

all attributes in *<grid>*			
all attributes in *type: visible*			Note that clip is always true for a gridlist.

<griddata> *visible, container*			Container element that treats its single child, or single rows-worth of children, as a template for the rest of the cells in the grid. If the special variable $$ appears in the definition of the child template, then it is replaced with the current scroll position in the set of values defined by the repeatover attribute. Only the number of children that fit in the grid are used, with the value of $$ being scrolled as focus is moved up and down the grid.
contains	contained by		
any *visible* any *listener*	any *container*		
attributes	type	default	
repeatover	*resource path*	*No default. Must be supplied.*	Specifies the set of values to use for the $$ variable in the child elements. If the resource path is a folder, then the list of resources found in that folder are used (in numeric order) for the set of values for $$. If the resource path is a file, then

			the file is treated as an index file that specifies a list of values for $$.
all attributes in *<grid>*			
all attributes in *type: visible*			

<image> *visible*			Draws an image.
contains	contained by		
any *listener*	any *container*		
attributes	type	default	
res *(modifiable)*	*resource path*	*none*	The resource path of the PNG file. Image is a transparent blank if res is not suppplied.
frames	*integer*	*1*	The number of frames (side by side images) in the PNG file. The image width is therefore the real PNG width divided by the number of frames.
index *(modifiable)*	*integer*	*1*	The frame number (counting from 1) to display.
all attributes in *type: visible*			The default for w/h is to shrink to fit the supplied image. If the image is not found, then w/h default as normal. If w/h are supplied,

			the image is aligned to the top left corner.

<tile> *visible*			Draws a tiled image. If borders are also supplied, the image is tiled by preserving corners and edges, tiling these lengthways as necessary.
contains	contained by		
any *listener*	any *container*		
attributes	type	default	
res *(modifiable)*	*resource path*	*none*	The resource path of the PNG file. The tile is transparent blank if the res is not supplied.
bdt	*integer*	*0*	The thickness of the top border. If zero, the tiling has no top edge tile.
bdl	*integer*	*0*	The thickness of the left border. If zero, the tiling has no left edge tile.
bdr	*integer*	*0*	The thickness of the right border. If zero, the tiling has no right edge tile.
bdb	*integer*	*0*	The thickness of the bottom border. If zero, the tiling has no bottom edge tile.
all attributes in *type: visible*			

<text> *visible*			Draws a text string. Text can be single or multiline, scrollable or not, editable or not. Text is drawn with device specific fonts.
contains	contained by		
any *listener*	any *container*		
attributes	**type**	**default**	
res *(modifiable)*	*resource path*	*none*	The resource path of the text string to display (initially if editable). A transparent blank is drawn if not supplied.
font	fixed serif sansserif system	serif	Device specific font.
size	small medium large	small	Device specific size. Should map to 9pt, 12pt and 18pt respectively.
weight	plain bold	plain	Device specific weight for the font.
slant	plain italic	plain	Device specific weight for the font.
align	left centre center right	left	The horizontal alignment of the text string inside the frame of the text box. There is no vertical alignment control, use the y attribute to control the text box position instead.
color, colour *(modifiable)*	*colour*	#ff000000 (black)	The colour of the text. If translucent alpha is not supported, the alpha component is rounded up to full opacity.

fxcolor, fxcolour *(modifiable)*	*colour*	*#00000000*	The colour of the text effect. The default text effect is a glow background.
multiline	*boolean*	*false*	If false, the string is drawn on a single line. The width of this element will default to the length required to exactly fit the string. If true, the string will be drawn on multiple lines. The width will default to be the same as the parent element. The height will default to the height required to exactly fit the number of lines for the string.
scrollable	*boolean*	*false*	If true, the view of the string can be scrolled (horizontally for single line, vertically for multiline) when this element has the focus. Focus is released when the end or beginning of the string is reached, or if a cursor key is pressed in the non-scrolling direction.
editable	*resource path to writable resource*	*None*	If supplied, this element is an editable text box. Text editing is drawn in a device specific way, and may involve pressing select to activate text editing. The edited value of the string

			is stored in the resource path supplied by this attribute.
all attributes in type: *visible*			

<throw> *listener*			Throws an event. Events can be sent to other parts of the content tree or to an actor.
contains	contained by		
<param>	any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only trigger when the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event when the parameter value is '_select'
event	*event name*	*none*	The name of the event to throw. If this is an Actor event, it will automatically be sent to the relevant Actor, regardless of the specified target. Use square brackets to specify an anonymous parameter value to accompany this event. Use <param> children to specify named parameters for this event.

			If the event is the 'focus' event, then this will cause the focus to move to the target element (within the layer of the target element).
target	*element ID*	_top	The element ID of the element to send this event to. If not supplied then _top is used. If the event is an Actor event, this attribute is ignored.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<att> *listener*			Modifies an attribute of its parent *visible* when switched on. <att> is switched on by the event specified in the when attribute. It is switched off by the '!' version of the event. If several <att>s modify the same parent attribute, the last <att> that is switched on wins.
contains	contained by		
	any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only switch on when the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event

			when the parameter value is '_select'
name	*attribute name*	*none*	The name of the attribute in the parent *visible* to modify. The attribute must be *modifiable* as indicated in the attribute boxes in this spec.
value	*same as attribute being modified*	*none*	The new value for the named attribute of the parent *visible*. Use the @-symbol to reference the value of a named parameter of the incoming event.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<anim> *listener*			Continuously modifies an attribute of its parent *visible* while switched on. The animation is started by the event, and restarted every time the event arrives subsequently. The modification (wherever the animation has got to) is switched off when the '|' version of the event arrives.
contains	contained by		
	any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only switch on when

			the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event when the parameter value is '_select'
name	*attribute name*	*none*	The name of the attribute in the parent *visible* to modify. The attribute must be *modifiable* as indicated in the attribute boxes in this spec.
startvalue	*same as attribute being modified*	*none*	The value to use at the start of the animation. If not supplied, the current value is used. The current value depends on all previous *listener* elements that modify the same attribute and the value specified by the parent *visible* itself.
endvalue	*same as attribute being modified*	*none*	The value to use at the end of the animation. This value is reached at the time specified by the duration attribute. If not supplied, the current value of the attribute is used in the same way as startvalue above.
duration	*integer number of milliseconds*	*300*	The length of time taken to animate the named attribute from startvalue to endvalue once. Note this is not the total duration of the animation which can be

			calculated by multiplying the number of repeats by this duration.
repeat	*integer* -1 =*forever*	0	The number of times to repeat the animation after the first time through, i.e. setting it to 1 will result in the animation being played twice.
bounce	*boolean*	false	If true, the animation will play backwards on alternate repeats.
persist	*boolean*	*depends...*	If true, the animation will hold the endvalue as the modification until switched off by the '!' event. If false, the animation will revert to the startvalue at the end of the animation and hold that value until the animation is switched off. The default depends whether the event is a normal event or a '!' version of an event. If the event is normal, the default is true. If the event is a '!' event, the default is false.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<load> listener			Loads some new content into the supplied target element.
contains <param>	contained by any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only trigger when the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event when the parameter value is '_select'
res	*resource path*	*none*	The resource path of the trigml file to load.
target	*element ID*	_top	The element ID to replace the children of.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<include>			Inlines the specified trigml file. The trigml in the file is treated as if it had been originally declared in place of this <include> element.
contains <param>	contained by any *element*		
attributes	type	default	
res	*resource path*	*none*	The resource path of the trigml file to include.

<param>			Supplies a parameter name and value to a <load>, <include> or <throw> element.
contains	contained by		
	<load> <include> <throw>		
attributes	type	default	
name	*parameter name*	*none*	The name of the parameter. The $-symbol is used to reference the parameter when used in a <load> or <include>. The @-symbol is used to reference the parameter when used with an event.
value	*value*	*none*	The value of the parameter.
valuefrom	*resource path*	*none*	The resource path of a file to read the contents of to obtain the value of this parameter.

<setvar> *listener*			Sets a variable. The variable can only used when loading new content. If <setvar> triggers on the 'entry' event, the variable cannot be used until the next <load> tag is used.
contains	contained by		
<param>	any *visible*		
attributes	type	default	
when	*event name and optional parameter value*	*none*	The event to listen for. If a parameter value is supplied in square brackets [], then this will only trigger when the event with that parameter value is received. E.g.: when="keypress[_select]" triggers on the keypress event when the parameter value is '_select'
name	*variable name*	*none*	The name of the variable.

value	*value*	*none*	The value to put in the variable. The variable can be referenced with the $-symbol in subsequent <load> actions.
consume	*boolean*	*false*	If true, the event propagation will stop at this element. No further *listeners* will trigger on the incoming event after this element.

<paintif> *container, visible*			Only paints its contents if the specified resource exists or the path is valid. The contents are still in the tree, and still respond to events, however, none of the contents are painted if the condition is not met. <paintif> can be used in place of group.
contains	contained by		
any *visible* any *listener*	any *element*		
attributes	type	default	
res	*resource path*	*none*	The resource path to test for the existence of.
isvalid	*resource path*	*none*	The resource path to test the validity (as a resource path) of. Note this will not actually check if the file exists, merely whether or not the path is a valid path. This is useful for testing whether $$ is in range or not.

<ticker> *visible*		Scrolls a series of items onto, then off, the frame of this element. The visible child element of <ticker> is used as a template for each item. Each item is scrolled on from below the element up into a centre-left-aligned
contains	contained by	
any *listener* any one *visible*	any *container*	

			position. The item is then paused before scrolling it off to the left. Use the $$ variable in the template to vary the item on each scroll past. The list is restarted at the top when the last item has been scrolled past.
attributes	type	default	
repeatover	*resource path*	*No default. Must be supplied.*	Specifies the set of values to use for the $$ variable in the child elements. If the resource path is a folder, then the list of resources found in that folder are used (in numeric order) for the set of values for $$. If the resource path is a file, then the file is treated as an index file that specifies a list of values for $$.
all attributes in type: *visible*			

<batterylevel> *visible*			Draws the battery level using the supplied image as a multi-framed image. The current value of the battery level is mapped onto the proportional frame number.
contains	contained by		
any *listener*	any *container*		
attributes	type	default	
res *(modifiable*	*resource path*	none	The resource path of the PNG file that holds all the states of the battery level.
frames	*integer*	*1*	The number of frames (side by side

			images) in the PNG file. The image width is therefore the real PNG width divided by the number of frames. The frame that is displayed depends on the current battery level.
all attributes in *type: visible*			The default for w/h is to shrink to fit the supplied image. If the image is not found, then w/h default as normal. If w/h are supplied, the image is aligned to the top left corner.

<signalstrength> *visible*			Draws the signal strength level using the supplied image as a multi-framed image. The current value of the signal strength level is mapped onto the proportional frame number.
contains	contained by		
any *listener*	any container		
attributes	type	default	
res *(modifiabl*	*resource path*	*none*	The resource path of the PNG file that holds all the states of the signal strength level.
frames	*integer*	*1*	The number of frames (side by side images) in the PNG file. The image width is therefore the real PNG width divided by the number of frames. The frame that is displayed depends on the current signal strength level.
all attributes in *type: visible*			The default for w/h is to shrink to fit the supplied image. If the image is not found, then w/h default as normal. If w/h are supplied, the image is aligned to

			the top left corner.

<phonestatus> *visible*			Draws a row of phone status icons. The icons are packed together, and are drawn, left to right, in the order specified in the include attribute. Use a blank image in order to reserve a space for an icon that is currently not visible.
contains	contained by		
any *listener*	any container		
attributes	type	default	
res *(modifiabl*	*resource path*	*none*	The root folder for the collection of icon images. For each capability specified by the include attribute, this element will look for a folder of the same name. Within that folder, this element will look for an image with a name equal to the current value of that capability.
include	*list of semi-colon separated capability names*	*none*	The names the status icons to display. Each name is a capability and should have a folder under the root folder specified by the res attribute.
all attributes in type: *visible*			

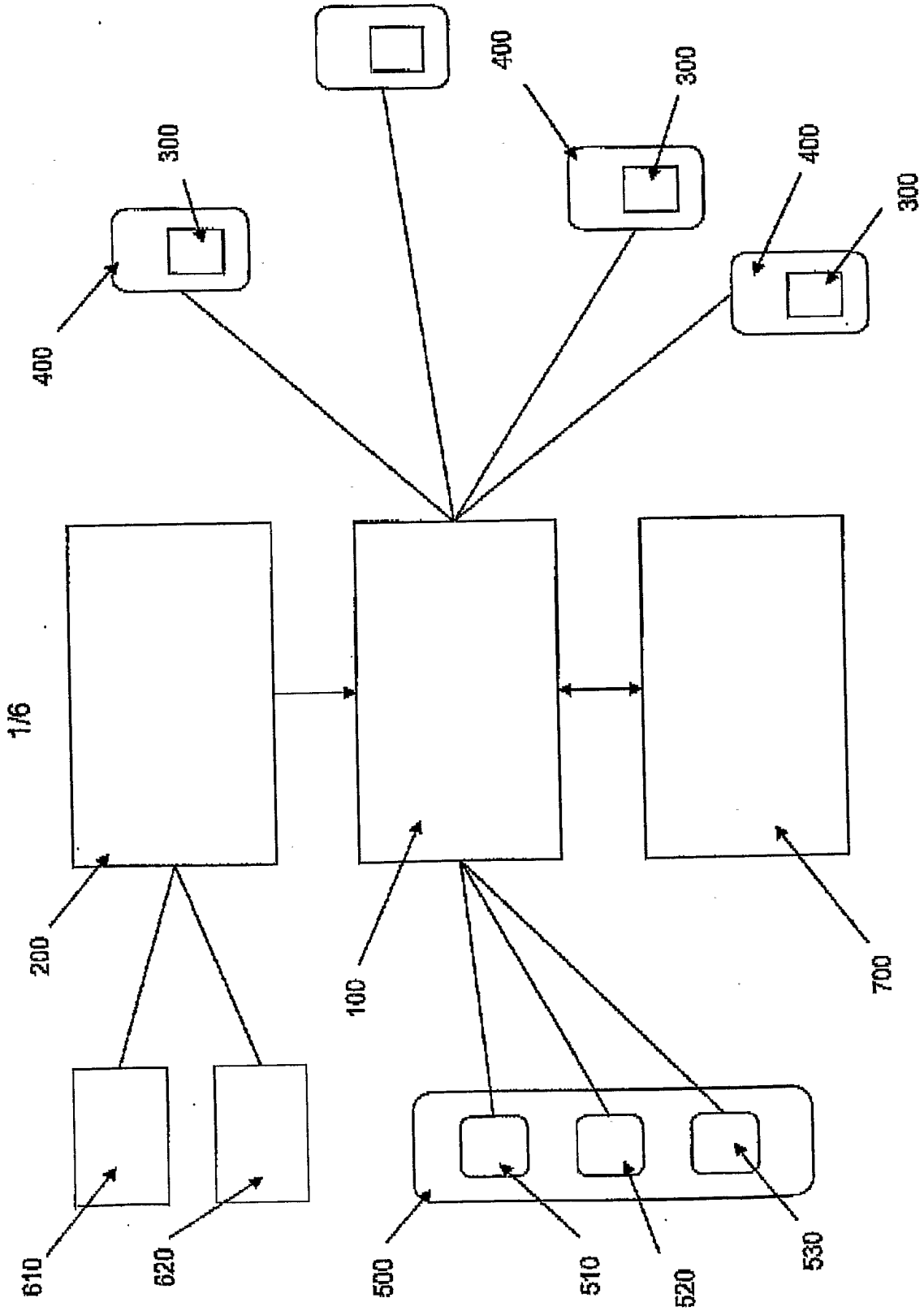


Figure 1

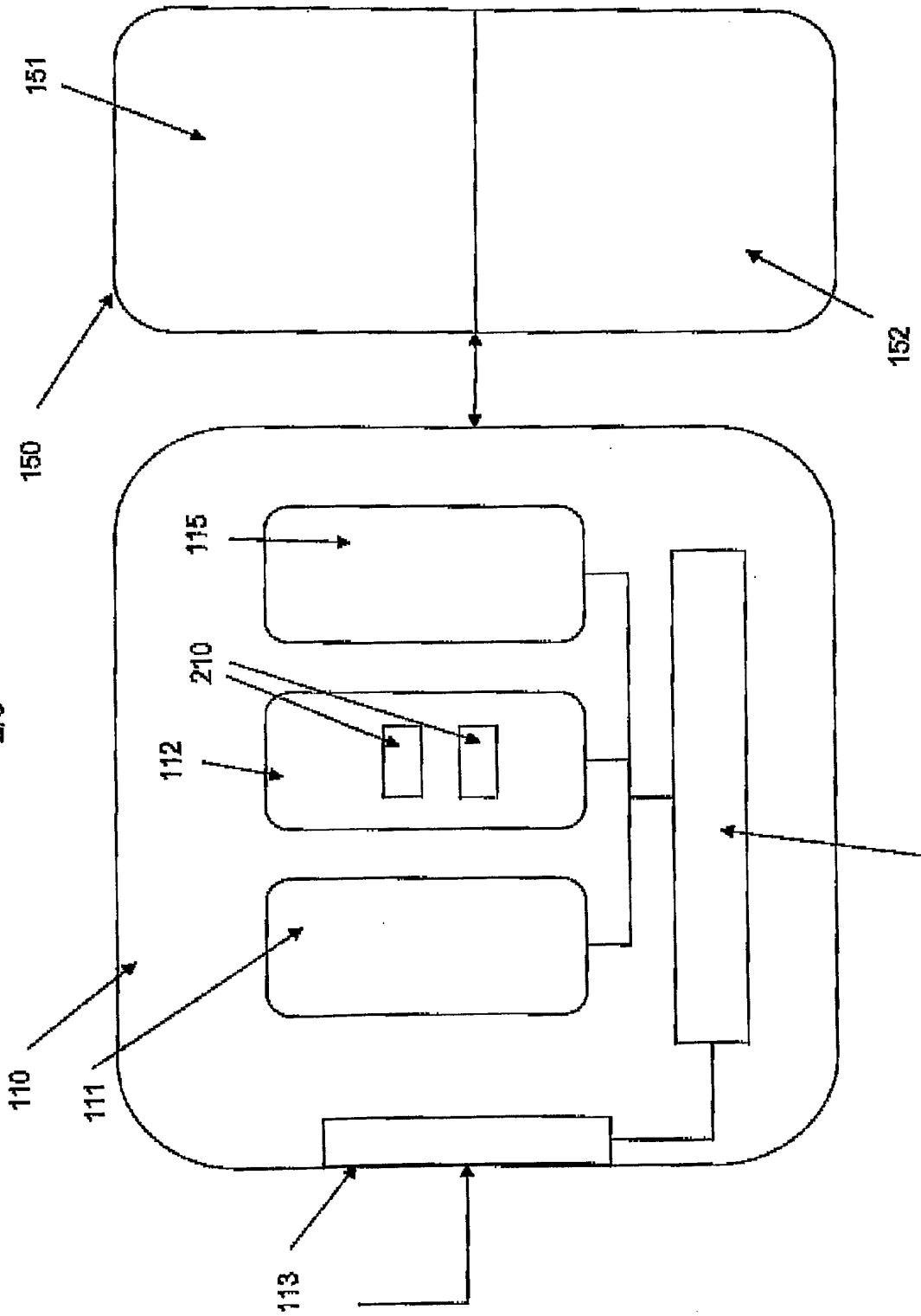


Figure 2

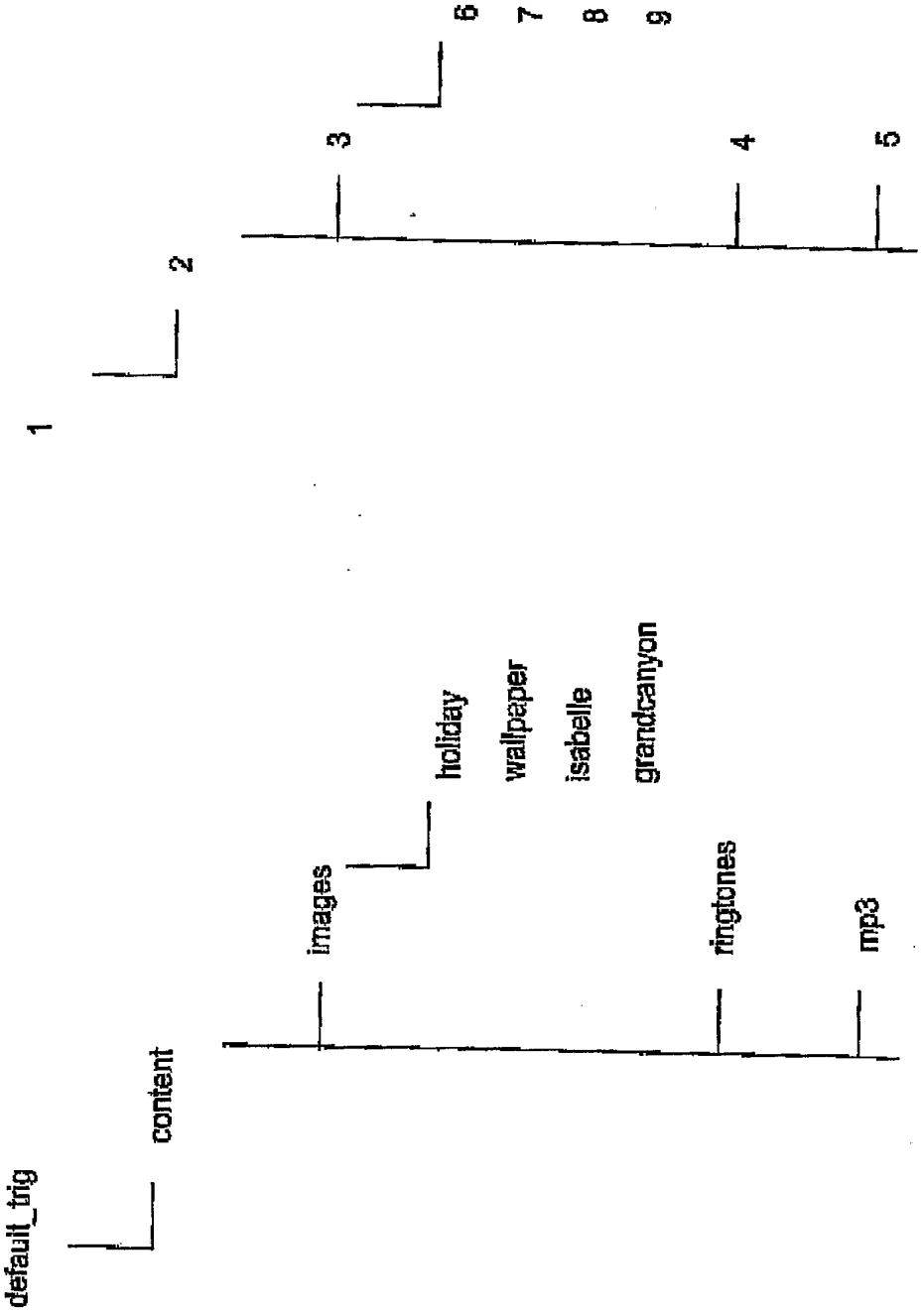


Figure 3a

Figure 3b

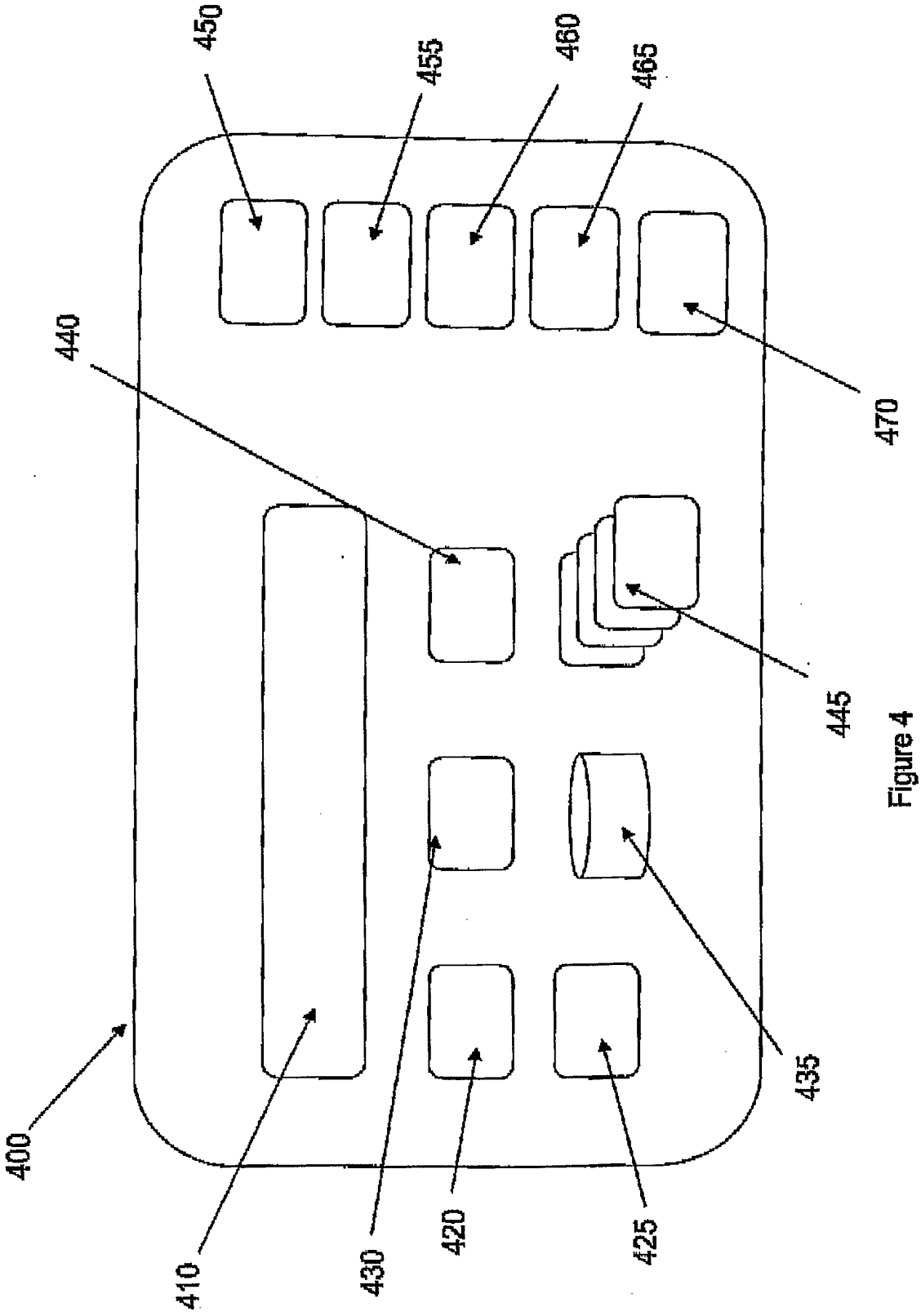
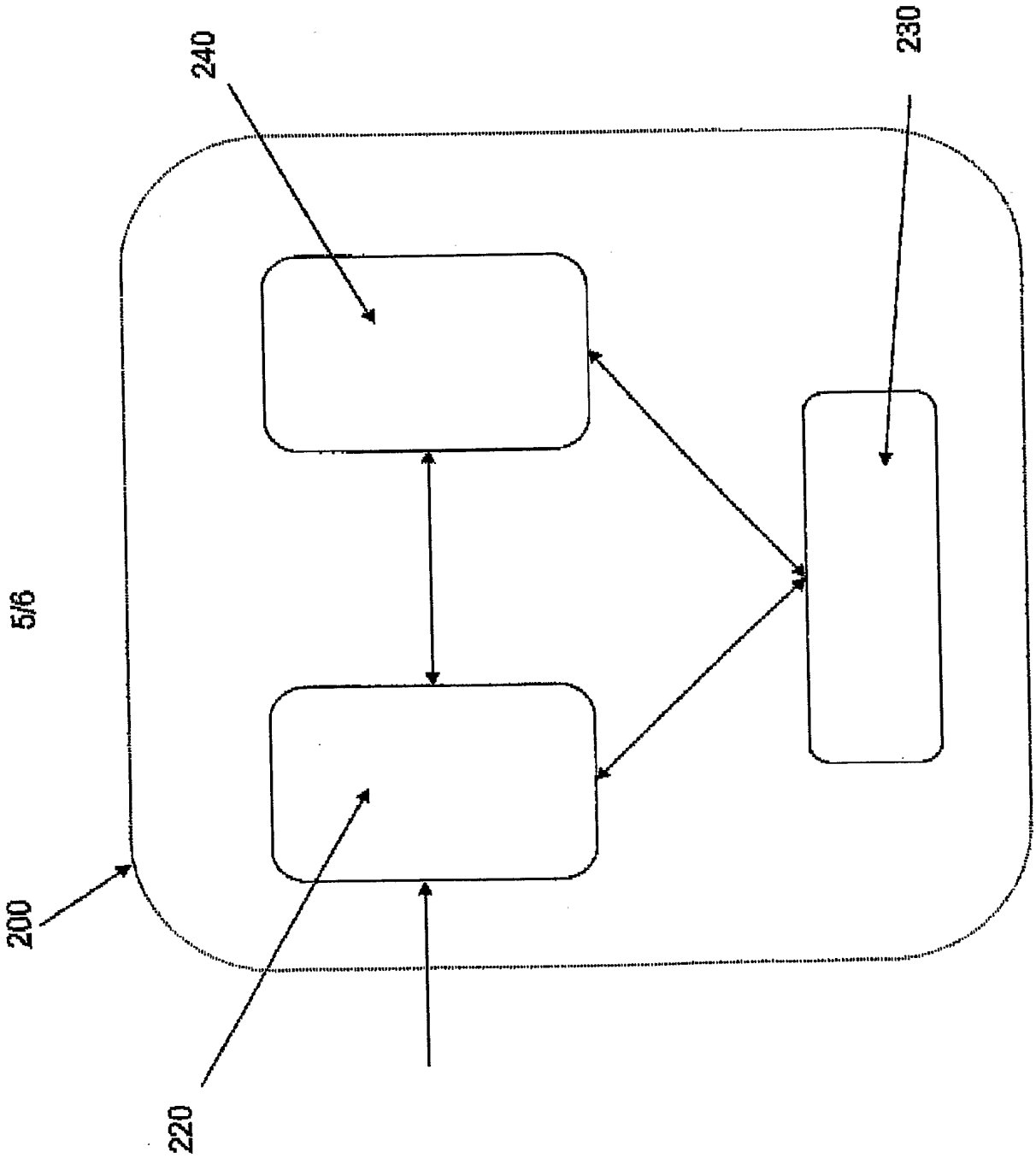


Figure 4

Figure 5



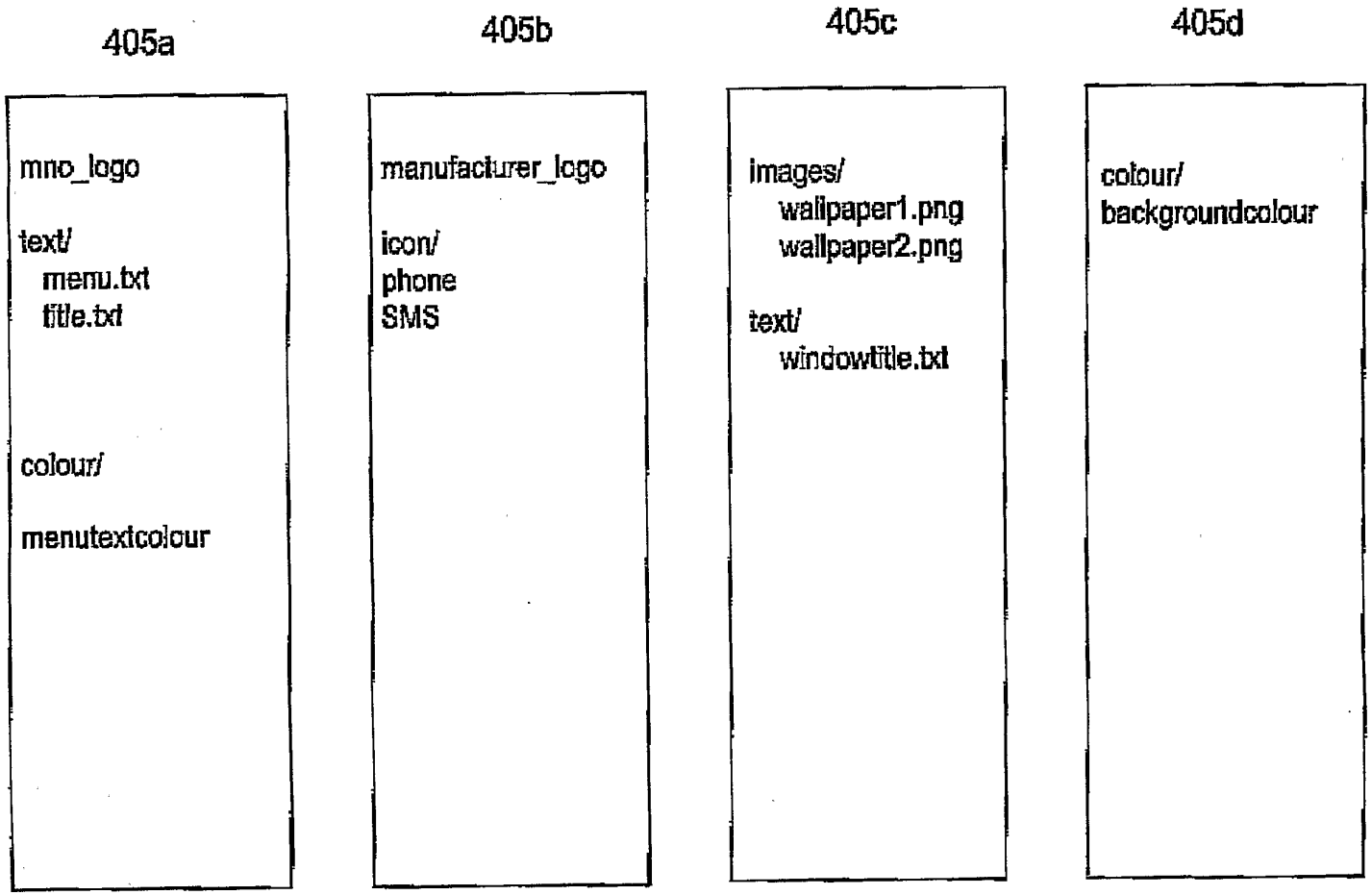


Figure 6